

1            SYSTEM AND METHOD FOR MEMORY FAILURE RECOVERY USING  
2                            LOCKSTEP PROCESSES

3  
4                            BACKGROUND OF THE INVENTION

5            1.     Field of the Invention

6            The present invention relates generally to systems and methods for memory  
7 failure recovery, and more particularly to memory failure recovery using lock-step  
8 processes.

9            2.     Discussion of Background Art

10           Memory Failure Recovery (MFR) describes an area within the general field of  
11 fault tolerant computer systems. Fault-tolerant computer systems or components  
12 incorporate backup hardware and/or software which are designed to be quickly  
13 brought on-line upon failure of primary hardware and software elements with minimal  
14 loss of service. Well known manufacturers of fault-tolerant systems and components  
15 include Compaq's Non-Stop product line, Marathon Technologies, and Stratus  
16 Computer.

17           Fault-tolerant techniques include periodically "check-pointing" critical data,  
18 duplexing selected hardware components, such as the microprocessors, mirroring  
19 disks, and "lock-stepping" multiple processors together. When a failure occurs, ideally  
20 the fault-tolerant system repairs itself often without even interrupting internal  
21 processes or computer users.

22           MFR techniques also include fault-tolerant systems for recovering from  
23 memory hardware errors. Three kinds of memory hardware errors exist: design  
24 errors, hard errors, and soft errors. Repair techniques for design and hard errors are  
25 almost always fatal unless protected against, and are typically limited to either  
26 refining the hardware's design or replacing an actual hardware component which

1 failed. However, as design techniques and hardware reliability have improved, design  
2 and hard errors have become a dwindling portion of memory hardware errors.

3 Instead, in matured and refined hardware systems soft errors are a growing  
4 and often the highest percentage of all three types of memory hardware errors. Soft  
5 errors occur on well designed and reliable hardware which has been affected by one  
6 or more unpredictable events in the operating environment. As examples, background  
7 radiation and cosmic rays can randomly and unpredictably interfere with memory  
8 hardware operation and/or corrupt data stored therein.

9  
10 Soft errors are a pointedly serious problem in low-profit margin Commodity  
11 Off The Shelf (COTS) systems. Such systems typically have very minimal, if any,  
12 hardware redundancy and/or error detection and correction systems, even though they  
13 are becoming ubiquitous tools within the office and home.

14 Mass marketed systems have two simple forms of support for memory soft  
15 errors. For several years memory systems have been available for commodity  
16 systems using parity or Error Correction Codes (ECC) to detect the presence of errors  
17 in memory and correct single bit errors. On error, systems either bring themselves to  
18 an abrupt halt or cause a severe signal in the processor. Low-cost processors, such as  
19 an Intel IA-32 and IA-64 processors, now contain this signaling support which is  
20 called a Machine Check Abort (MCA) exception. On the detection of an error, this  
21 severe error typically leads to a system halt performed by the operating system. Error  
22 correction codes are effective for detecting errors and correcting the simplest errors,  
23 however, the fore mentioned system does not cater for recovery from errors when  
24 they do occur and cannot be corrected in hardware.

25

1           Figure 1 is a data-flow diagram of memory failure within such a IA-64 COTS  
2 computer system 100. A typical IA-64 computer system 100 includes a kernel  
3 process 102 in communication with a large number of other computer processes, such  
4 as process 104, over an input-output (I/O) channel 106. In response to a soft memory  
5 error 108 which corrupts process 104, the kernel 102 generates an MCA signal 110  
6 which typically requires that process 104 be terminated. If process 104 served an  
7 application or other some other top-level program, or utility, such programs or utilities  
8 will then terminate, perhaps resulting in a substantial loss of important data which had  
9 not yet been saved. Even worse, process 104 could have been a key operating system  
10 process which causes a system crash, requiring that the whole computer be rebooted.  
11 Such a drastic action not only results in a loss of important data and perhaps  
12 termination of network communications, but also results in a significant loss of time  
13 to the computer's 100 users, who must not only reboot the computer, but also bring up  
14 the application programs again and perhaps re-enter data.

15  
16           Lock-step processors, mentioned above, are one approach toward  
17 implementing a fault-tolerant computing systems which can perhaps recover from  
18 some design and hard errors. Lock-step processors are found within Compaq  
19 Himalayas Non-Stop Series of computers and IBM's S/390 computer series. Lock-  
20 step processor systems include two hardware processors strictly synchronized cycle-  
21 by-cycle. They execute exactly the same instruction each cycle. Lock-step systems  
22 also include a substantial amount of internal circuitry inside each of the processors for  
23 internally checking that the two lock-stepped processors are indeed operating  
24 consistently. Lock-step processors, however, are still vulnerable to memory hard and  
25 soft errors since the two processors share memory resources. Thus, if the shared  
26 memory fails, the lock-step processors will not be able to recover can not recover and

1 the computer must be rebooted. Even further, lock-step processor systems are very  
2 expensive, since duplication of very expensive and necessarily complex circuitry is  
3 required.

4

5 Another approach toward fault-tolerant computing employs fail-over clusters.  
6 A fail-over cluster consists of at least two interconnected nodes/computers. The two  
7 nodes rely on intercommunication of shared data for recovery support. During  
8 normal operation, the two nodes share a predetermined portion of all processing tasks.  
9 Upon failure of one of the nodes, however, the other node assumes responsibility for  
10 all processing tasks. Such clusters also suffer from the same cost and complexity  
11 limitations, due to the node duplication required. Furthermore, upon a failure  
12 condition in such clusters, all processing tasks are switched over to the other  
13 node/computer, which may not always be a desirable situation due to the high load.

14

15 As a final example, Cornell University has developed a fault-tolerant  
16 computing technique based on "Hyper-Visors." A Hyper-Visor is a software virtual  
17 machine that is instantiated between a computer's processor and the computer's  
18 operating system, and gives the illusion of multiple processors on one processor. In a  
19 typical fault-tolerant Hyper-Visor implementation, the processor hosting a copy of the  
20 Hyper-Visor, is part of a complete system, but gives the Hyper-Visor gives the  
21 illusion of multiple processors sharing the rest of the system. The Hyper-Visor  
22 implements two or more processors, each of which is able to run its own operating  
23 system, application program, and utility processes. During normal operation, only the  
24 first virtual processor interacts with system software and resources. Upon a failure on  
25 the first virtual processor, however, the backup virtual processor takes over and  
26 processing continues. Like the fail-over cluster technique, all application jobs are

1 switched over to the other Hyper-Visor processor. However, since virtual processors  
2 are sharing resources, such as memory and disks, errors in these may affect both  
3 virtual machines. Lastly, virtual machines must present a fault isolation boundary to  
4 be effective for fail-over support. Unfortunately, this requires hardware support for  
5 the virtual machine monitor and critical system errors such as memory errors may not  
6 be isolatable.

7

8 In response to the concerns discussed above, what is needed is a system and  
9 method for memory failure recovery that overcomes the problems of the prior art.

10

## SUMMARY OF THE INVENTION

The present invention is a system and method for memory failure recovery using lock-step processes. The method of the present invention includes the steps of maintaining a predetermined number of duplicate and primary processes; keeping the processes in synchronization; managing the processes so that a single process image is presented to an external environment; detecting a computer system exception which affects one of the processes; and terminating the affected process.

Within the system of the present invention, a primary process memory space hosts a primary process; a duplicate process memory space hosts a duplicate process corresponding to the primary process; a synchronization buffer for keeps the duplicate process in synchronization with the primary process; a processor generates an exception signal in response to detection of a memory failure condition which affects the primary process; and an operating system receives the exception signal, terminates the affected primary process, and maintains a predetermined number of primary and duplicate processes.

The system and method of the present invention are particularly advantageous over the prior art because the present invention enables dynamic fault-tolerance adjustment without added hardware expense and can be implemented in standard Commodity Off The Shelf (COTS) systems.

These and other aspects of the invention will be recognized by those skilled in the art upon review of the detailed description, drawings, and claims set forth below.

BRIEF DESCRIPTION OF THE DRAWINGS

1  
2 Figure 1 is a data-flow diagram of memory failure within a Commodity Off  
3 The Shelf (COTS) computer system;  
4 Figure 2 is a data-flow diagram of a first embodiment of a system for lock-step  
5 process memory failure recovery within a computer;  
6 Figure 3 is a fault-tolerance level data structure for dynamically specifying a  
7 fault-tolerance level for primary computer processes operating within the computer;  
8 Figures 4A & 4B together are a method for lock-step process memory failure  
9 recovery;  
10 Figure 5 shows a second fault-tolerance level data structure for a second  
11 embodiment of the system; and  
12 Figure 6 is a data-flow diagram showing the second embodiment of the  
13 system.  
14

1                    DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

2                    Figure 2 is a data-flow diagram of a first embodiment of a system 200 for  
3 lock-step process memory failure recovery within a computer 202. Figure 3 is a fault-  
4 tolerance level data structure 300 for dynamically specifying a fault-tolerance level  
5 for primary computer processes 302 operating within the computer 202. Figures 4A  
6 & 4B together are a method 400 for lock-step process memory failure recovery.  
7 Figures 2, 3 and 4 are discussed together.

8                    The computer 202 is under control of an Operating System (OS), which  
9 includes a kernel process 204. The computer 202 also hosts a large number of  
10 processes (not shown) which provide services to the operating system, application  
11 programs, computer utilities, and almost all other computer functionality.

12                   The method 400 begins in step 402 the OS associates a fault tolerance variable  
13 with each process within a set of primary processes operable within the computer 202.  
14 The set of primary processes includes those processes which are currently identified  
15 by the operating system as in an active state and in service of computer system  
16 functionality. The primary processes also includes those that will be created when an  
17 application program is launched. Computer system functionality which is external to  
18 such processes is herein defined as the processes's environment. The primary  
19 processes may be either parent or child processes.

20                   The system 200 shows only one primary process P0 206 for the purposes of  
21 this discussion; however, in a typical implementation of the present invention there  
22 will likely be hundreds of primary processes which are active at any one time.



1           Next, in step 404 values are assigned to each fault-tolerance variable 304, in  
2   response to either predetermined default values, dynamically specified system  
3   administrator selected values, or application program specified values which are  
4   stored in the fault-tolerance level data structure 300 of Figure 3. For example, with  
5   respect to system 200, the primary process P0 206 has the value of its fault tolerance  
6   variable 304 set to "2".

7           In step 406, the OS retrieves the value of the fault-tolerance variable 304  
8   corresponding to a primary process within the set of primary processes 302. Next in  
9   step 408, the OS sets a number of duplicate processes equal to the value of the fault-  
10   tolerance variable 304 of the primary process. The setting process in step 408 may  
11   result in the creation of new duplicate process, the termination of an excessive number  
12   of duplicate processes, or maintenance of a current number of duplicate processes,  
13   depending upon the current value of the fault-tolerance variable 304 and how many  
14   duplicate processes currently exist. Preferably, new duplicate processes are created  
15   immediately after corresponding primary processes are created. In system 200, since  
16   the value of the fault tolerance variable 304 is "2" for primary process 206, the OS  
17   creates two duplicate processes P0' 208 and P0'' 210.

18           Also since different fault-tolerance values may be assigned to either parent or  
19   child primary processes, some implementations of the present invention may have a  
20   parent primary process with only one duplicate, but a corresponding child primary  
21   process with three or more duplicates. Alternatively, the child primary processes can  
22   have few duplicate processes than a corresponding parent primary process.

23           In step 410, the OS allocates a new memory space within the computer's 202  
24   memory hardware (not shown) to each of the duplicate processes. The new memory  
25   space is preferably separate from a primary memory space allocated to the primary  
26   process. By keeping the primary and duplicate process memory spaces separate the

1 present invention protects a computer from memory failure errors occurring in the  
2 memory space allocated to the primary process. Thus in system 200, primary process  
3 P0 206 has its own dedicated memory space, and duplicate processes P0' 208 and  
4 P0'' 210 each have their own separate memory spaces respectively. By increasing the  
5 number of duplicate processes, a systems administrator can protect the computer 202  
6 from any number of simultaneous soft memory errors, depending upon how critical  
7 the corresponding primary process is to either OS, application program, or utility  
8 program functioning.

9 For example, if the computer 202 functioned as a server on a network, a  
10 systems administrator may specify multiple duplicate processes for all computer 202  
11 primary processes. Wherein, if the computer 202 functions as a stand alone system,  
12 perhaps the systems administrator or a user would create duplicate processes only for  
13 the OS or certain key application programs.

14 The system 200 includes a synchronization buffer 214 through which the  
15 primary process P0 206 and the duplicate processes P0' 208 and P0'' 210 maintain  
16 communication with the kernel process 204 and thus the external environment. All  
17 these processes are linked to the synchronization buffer 214 though I/O channels 212,  
18 216, 218, and 220 as shown. The synchronization buffer 214 is under control of a  
19 buffer controller 221. The buffer controller 221 permits both the primary and  
20 duplicate processes to receive data or signals from the external environment.  
21 In this way, as noted in step 412, the duplicate and primary processes are kept in  
22 synchronization in response to interactions with the external environment.

23 In contrast however, the buffer controller 221 preferably permits only one of  
24 the processes 206, 208 or 210 to transmit a response, such as commands, system calls,  
25 library calls and the like, out of the synchronization buffer 214 over I/O channel 212  
26 back to the external environment. All other responses from the other processes 206,

1 208, or 210 are masked within the synchronization buffer 214 by the buffer controller  
2 221 and thus are not transmitted back to the kernel process 204 over I/O channel 212.  
3 Many different process selection criteria may be used to determine which of the  
4 processes 206, 208, or 210 is permitted to respond. Preferably the process which  
5 responds most quickly is permitted to respond.

6 However, the processes are also synchronized when one of the processes  
7 transmits a response. Thus in a preferred embodiment of the present invention, both  
8 the primary and duplicate processes operate in a loosely-coupled lock-step. Loosely-  
9 coupled means herein that the primary and duplicate processes are preferably  
10 synchronized only upon receipt of data or signals from the external environment, or  
11 when commands, system calls, library calls and the like are sent to the external  
12 environment.

13 Those skilled in the art however, will recognize that other systems and  
14 methods for keeping the duplicate processes in synchronization with the primary  
15 process may also be employed. In fact, an exact method by which the processes are  
16 kept in synchronization is preferably left to the discretion of the systems administrator  
17 presiding over a particular implementation of the present invention. Such alternative  
18 synchronization methods may be based on timing concerns, such as to minimize  
19 processor time spent performing synchronization, or based on synchronization  
20 overhead concerns, such as by looking for windows of relative processor inactivity  
21 during which to perform synchronization.

22 In this way, as noted in step 414, only a single process image is presented to  
23 the external environment. The masked out primary and/or duplicate processes can  
24 thus be thought of as black-boxes during normal system 200 operation.

25

1 In step 416, steps 406 through 414 are repeated for all remaining primary  
2 process in the set of primary processes. Next in step 418, the method 400 returns to  
3 step 406 in response to input from the system administrator or another source, which  
4 changes the value of the fault-tolerance variable 304 for any process in the set of  
5 primary processes.

6 Thus, the present invention along with the fault-tolerance level data structure  
7 300 gives users and systems administrators an ability to, dynamically or by default,  
8 assign a unique fault-tolerance level (a.k.a. a High Availability (HA) level) to each  
9 and every primary process operating on a computer system. The present invention  
10 and data structure 300 also permit fault-tolerance levels to be modified during  
11 computer 200 operation without having to terminate application programs or reboot  
12 the computer 200. Thus for example, if the system administrator observes that  
13 memory errors tend to be less frequent in the kernel process's 204 memory space  
14 when compared with a memory space allocated to a user application program, the  
15 system administrator can merely change the value of the fault-tolerance variable 304  
16 for certain processes servicing the user application program. The present invention's  
17 fault-tolerance technique is thus much more flexible and requires less complex  
18 hardware than prior art techniques.

19  
20 While preceding paragraphs have discussed how preparation for the present  
21 invention's system and method for memory failure recovery using lock-step processes  
22 is implemented, the paragraphs to follow discuss how the system and method  
23 responds to an actual memory failure condition.

24  
25 In step 420, the OS has just detected a computer system exception in response  
26 to some sort of failure condition. The failure condition may be of any type which

1 affects operation of one or more primary or duplicate processes within the computer  
2 202. While memory failures are contemplated as a main source for such exceptions,  
3 other non-memory failure conditions may also corrupt one or more processes.  
4 Detection may occur in any number of ways, one of which is shown in Figure 2,  
5 whereby a computer processor (not shown), hosting the kernel and other processes,  
6 generates a Machine Check Abort (MCA) exception signal 222, upon detection of a  
7 fatal hardware error, which can not be corrected by either hardware or firmware.

8       Next in step 422, the OS identifies all primary and/or duplicate processes  
9 corrupted by the failure condition. In the system 200 example, only primary process  
10 P0 206 has affected by a failure condition 224. In response to the failure condition,  
11 all corrupted primary and duplicate processes are terminated in step 424. Thus in the  
12 example only primary process P0 206 is terminated.

13       Since the synchronization buffer 214 presents the external environment with a  
14 single “process image” and duplicate processes 208 and 210 can still respond to the  
15 external environment, termination of the primary process 206 is not detectable by the  
16 external environment, and thus application programs, computer system utilities or  
17 other computer functionality relying upon the terminated primary process need not be  
18 shut down and/or rebooted in response to the failure condition.

19       In step 426, the OS restores the total number of processes to the value of the  
20 corresponding fault-tolerance variable by returning to Step 408. And, in step 428, the  
21 OS puts the primary and duplicate processes back in computer’s process queue, after  
22 which process execution continues as if the failure-condition never occurred. After  
23 step 430 the method ends.

24

1           Figure 5 shows a second fault-tolerance level data structure 500 for a second  
2   embodiment of the system 200, and Figure 6 is a data-flow diagram 600 showing the  
3   second embodiment of the system 200. Figures 5 and 6 are discussed together.

4           The second fault-tolerance level data structure 500 identifies three primary  
5   processes 302 and values for their corresponding fault-tolerance variables 304. The  
6   processes include: a primary parent process P0 502 having the value of its fault  
7   tolerance variable set to “2”, a primary child process P00 504 having the value of its  
8   fault tolerance variable set to “3”, and a primary child process P01 506 having the  
9   value of its fault tolerance variable set to “1”.

10          As shown by the second data structure 500, child processes may have their  
11   fault-tolerance variable 304 set to a value different than their corresponding parent  
12   processes. For example, a primary parent process with one duplicate may have a  
13   primary child process having no duplicates. Thus the duplicate parent process will be  
14   kept in synchronization with the primary parent process while the primary child  
15   process will have no duplicate process to be kept in sync with. Alternatively, a  
16   primary parent process with one duplicate can have a primary child process with two  
17   duplicates.

18          Data-flow diagram 600 shows how duplicate processes corresponding to  
19   primary parent and child processes P0 502, P00 504, and P01 506 are kept in  
20   synchronization. Two duplicate parent processes P0' 602 and P0'' 604 have been  
21   created by the OS to backup primary parent process P0 502, since the value of  
22   primary parent process's P0 502 fault-tolerance variable was set to “2”. Duplicate  
23   processes P0' 602 and P0'' 604 of primary parent process P0 502 are kept in  
24   synchronization by routing all external communications sent and/or received over I/O  
25   channel 606 through synchronization buffer (P0) 608 in the same way as discussed  
26   with respect to Figure 2.

1 Similarly, three duplicate child processes P00' 610, P00'' 612, and P00''' 614  
2 have been created by the OS to backup primary child process P00 504, since the value  
3 of primary child process's P00 504 fault-tolerance variable was set to "3". Duplicate  
4 processes P00' 610, P00'' 612, and P00''' 614 of primary child process P00 504 are  
5 kept in synchronization by routing all external communications sent and/or received  
6 over I/O channel 616 through synchronization buffer (P00) 618.

7 And lastly, only one duplicate child process P01' 620 has been created by the  
8 OS to backup primary child process P01 506, since primary child process's P01 506  
9 fault-tolerance variable was set to "1". Duplicate process P01' 620 of primary child  
10 process P01 506 is kept in synchronization by routing all external communications  
11 sent and/or received over I/O channel 622 through synchronization buffer (P01) 624.

12 As mentioned above, those skilled in the art will recognize however that  
13 synchronization can be performed in many other ways and using different hardware  
14 than shown as well.

15  
16 While one or more embodiments of the present invention have been described,  
17 those skilled in the art will recognize that various modifications may be made.  
18 Variations upon and modifications to these embodiments are provided by the present  
19 invention, which is limited only by the following claims.